C++ code 2.4.12: Implementation of `computeA()` → GitLab

```
template <typename FUNCTOR1>
std::vector<Eigen::Triplet<double>> computeA(const Eigen::VectorXd &mesh,
                                             FUNCTOR1 &&alpha) {
  // Nodes are indexed as 0=x_0 < x_1 < ... < x_N = 1
  unsigned N = mesh.size() - 1;
  // Initializing the vector of triplets whose size corresponds to the
  // number of entries in a (N+1) x (N+1) tridiagonal (band) matrix
  std::vector<Eigen::Triplet<double>> triplets;
  // Maximal size of vector; avoids intermediate allocations
  triplets.reserve(3 * N + 1);
  // Some auxiliary variables
  double diag, off_diag;     // matrix entries
  double dx_left, dx_right;  // cell widths

  /* Computing diagonal entries */
  // First diagonal entry (left boundary node)
  dx_right = mesh(1) - mesh(0);
  diag = alpha((mesh(1) + mesh(0)) / 2.) / dx_right;
  triplets.push_back(Eigen::Triplet<double>(0, 0, diag));
  // Last diagonal entry (right boundary node)
  dx_left = mesh(N) - mesh(N - 1);
  diag = alpha((mesh(N) + mesh(N - 1)) / 2.) / dx_left;
  triplets.push_back(Eigen::Triplet<double>(N, N, diag));
  // All diagonal entries associated to interior nodes
  for (unsigned i = 1; i < N; ++i) {
    dx_left = mesh(i) - mesh(i - 1);
    dx_right = mesh(i + 1) - mesh(i);
    diag = alpha((mesh(i - 1) + mesh(i)) / 2.) / dx_left +
           alpha((mesh(i) + mesh(i + 1)) / 2.) / dx_right;
    triplets.push_back(Eigen::Triplet<double>(i, i, diag));
  }

  /* Computing off-diagonal entries */
  for (unsigned i = 0; i < N; ++i) {
    dx_right = mesh(i + 1) - mesh(i);
    off_diag = -alpha((mesh(i) + mesh(i + 1)) / 2.) / dx_right;
    triplets.push_back(Eigen::Triplet<double>(i + 1, i, off_diag));
    triplets.push_back(Eigen::Triplet<double>(i, i + 1, off_diag));
  }
  return triplets;
}  // computeA
```